Dynamic Programming: Problem 1

An M-matrix of size FxC is available (F is the number of rows and C the number of columns), whose cells have a positive integer numeric value. For example, the matrix with 4 rows and 5 columns below:

3	2	3	2	5
1	2	4	5	6
4	3	6	2	4
2	5	3	4	3

A movement between the squares M_{ij} and M_{pq} is valid only if (p=i && q=j+1) or if (p=i+1 && q=j). The path of the matrix is defined as the sequence of movements that lead from box M_{11} to the box M_{FC} and the cost of a path is equal to the sum of the values of the boxes that it crosses.

Design a Dynamic Programming algorithm that obtains the lowest cost of all the paths of a given matrix.

Dynamic Programming: Problem 1

- Content of the matrix: M[NF][NC]
- Only movements that advance in a row or a column are valid
- The cost of each movement corresponds to the sum of the least expensive option: advance in a row or in a column
- The cost associated to each position of the original matrix is stored: C[NF][NC]
- To know the movements, the auxiliar matrix is inspected from the end, once completely calculated: Mov[1…2][1…NF+NC-1]

Dynamic Programming: Problem 1

- NF = 3, NC =4

M

5	2	6	4
3	3	8	1
2	7	4	8

C

5	7	13	17
8	10	18	18
10	17	21	26

- Minimum cost: 26

- Minimum path:
  - Necessary movements: NMOV = NF + NC – 2 = 5
  - Traveled cells: NCells = NF + NC – 1 = 6
  - Minimum path: (1,1)→(1,2)→(1,3)→(1,4)→(4,2)→(4,3)

Dynamic Programming: Problem 1

- NF = 4, NC =5

M

3	2	3	2	5
1	2	4	5	6
4	3	6	2	4
2	5	3	4	3

C

3	5	8	10	15
4	6	10	15	21
8	9	15	17	21
10	14	17	21	24

- Minimum cost: 24

- Minimum path:

  - Necessary movements: NMOV = NF + NC – 2 = 7
  - Traveled cells: NCells = NF + NC – 1 = 8
  - Minimum path: (1,1)→(1,2)→(1,3)→(1,4)→(2,4)→(3,4)→(3,5)→(4,5)

Dynamic Programming: Problem 1

- How to fill the auxiliar cost matrix: C[i][j]
  - Inicialization of the first cell:
    - C [1] [1] = M [1] [1]
  - Inicialization of the first column:
    - C [i] [1] = M [i] [1] + C [i–1] [1] $\forall$ i
  - Inicialization of the first row:
    - C [1] [j] = M [1] [j] + C [1] [j–1] $\forall$ j
  - Rest of the matrix:
    - C [i] [j] = M [i] [j] + mínimo { C [i–1] [j], C [i] [j–1] } $\forall$ i > 1, j > 1

Dynamic Programming: Problem 1

```
const NF, NC = ...
types matrix= array[1... NF] [1...NC] of integer
types movements= array[1... 2] [NF+NC-1] of integer //To store the movements
fun CostsMatrix (I M: matrix; I/O C: matrix, I/O Mov: movements) ret Cost: integer
    var i, j, nmov: integer
    var M: matrix
    C [1] [1] = M [1] [1]
    for i=2 to NF do   C [ i ] [1] = M [ i ] [1] + C [ i - 1] [1]  efor //First column
    for j=2 to NC do   C [1] [ j ] = M [1] [ j ] + C [1] [ j - 1]  efor //First row
    for i=2 to NF do //Rest of the matrix
        for j=2 to NC do
            C [ i ] [ j ] = M [ i ] [ j ] + Mínimum { C [ i - 1] [ j ] , C [ i ] [ j - 1] }
        efor
    efor
    Mov [1] [NF+NC-1] = NF ; Mov [2] [NF+NC-1] = NC  ; i = NF; j = NC; //The last cell is (NF,NC)
    for nmov = NF+NC-2 to 1, with nmov = nmov – 1 //Movements from (NF,NC) backwards
        if  ( ( C [ i – 1] [ j ] < = C [ i ] [ j – 1] ) and ( i > 1) ) then
            i = i – 1
        else if ( ( i ==1 ) or ( j > 1 ) ) then
            j = j – 1
        eif
        Mov [1] [ nmov ] = i ; Mov [2][ nmov ] = j //Store the cells associated with the movements
    efor
    return  C [NF] [NC]
efun
```